Partial Rank Modulation for Flash Memories

Zhiying Wang
Electrical Engineering Department
California Institute of Technology
Pasadena, CA 91125, USA
Email: zhiying@caltech.edu

Jehoshua Bruck
Electrical Engineering Department
California Institute of Technology
Pasadena, CA 91125, USA
Email: bruck@caltech.edu

***Abstract*—Rank modulation was recently proposed as an information representation for multilevel flash memories, using permutations or ranks of n flash cells. The current decoding process finds the cell with the i-th highest charge level at iteration i, for $i = 1, 2, \ldots, n-1$. Motivated by the need to reduce the number of such iterations, we consider k-partial permutations, where only the highest k cell levels are considered for information representation. We propose a generalization of Gray codes for k-partial permutations such that information is updated efficiently.**

I. Introduction

Rank modulation was recently proposed as a robust and update-efficient information representation scheme for multilevel flash memories [1]. In rank modulation of n flash cells, information is stored as the rank of the cells. Let $c_1, c_2, \ldots, c_n$ be the distinct analog values of electric charge in these n cells and suppose $c_{p_1} > c_{p_2} > \cdots > c_{p_n}$. Then the induced rank vector is $(p_1, p_2, \ldots, p_n)$. For example, if $n = 5$, $(c_1, \ldots, c_5) = (0.3, 1.2, 1.0, 0.25, 0.5)$, then the rank is $(p_1, \ldots, p_5) = (2, 3, 5, 1, 4)$.

In flash memory, to decrease the charge level of a cell, we have to first erase a complete block of cells (containing about 10^5 cells [2]), which not only costs time and energy, but also reduces the lifetime of the device. In order to avoid overshooting when we increase the charge level, iterative injection is used in practice. The main advantage of using ranks over absolute values is that there is no need to charge the cells accurately. Each write operation only pushes one cell level to be the highest among the n cells, so there is no risk of overshooting. We call this writing method *push-to-the-top* operation. Thus the writing process is accelerated and higher reliability is obtained. In addition, leakage (or deflation) causes charge loss among all cells simultaneously, though not evenly, which will facilitate more write operations.

Since information is represented by permutations of charge levels of the flash cells, the decoding process involves sorting of the levels - not a straightforward operation if implemented in hardware. The current proposal for a decoding architecture for rank modulation is based on an iterative process involving (1) maximal level identification: identifying the cell with the maximal level among the remaining cells and (2) candidates update: excluding the current maximal cell from future maximal level identification. Namely, for n flash cells the decoding process has $n-1$ iterations.

Each iteration costs time, power, and circuit complexity. We would like to lower this cost while maintaining reasonable information capacity. Hence, we consider a generalized form of rank modulation, where information is represented by a permutation on the k largest values out of the n cells, for some $1 \leq k \leq n-1$. We call it *k-partial rank modulation* or *k-partial permutation*. The amount of information is $\log \binom{n}{k} k!$ bits and decoding takes k iterations. When $k = n-1$, it reduces to the original rank modulation. If we preserve the push-to-the-top write operations, we will get high speed and low error rate as in the original rank modulation. For example, when $n = 5$ and $k = 2$, we can represent $5 \times 4 = 20$ values while requiring 2 decoding iterations. On the other hand, when $k = n-1 = 4$, we can represent $5! = 120$ values with 4 iterations. The permutation $(2, 3, 5, 1, 4)$ induces the 2-partial permutation $(2, 3)$, and we will denote the permutation by $(2, 3|5, 1, 4)$ to emphasize the first 2 elements. If we push the third highest cell (cell 5) to the top, we get $(5, 2|3, 1, 4)$.

Consider every n cells as one logical digit, taking an alphabet size of $\binom{n}{k} k!$. Suppose we have a cycle that traverses the $\binom{n}{k} k!$ partial permutations, and increasing a logical digit by one corresponds to moving forward one step in the cycle. Then we can take a set of such logic digits, and increase some digits by a small amount at a time (e.g., floating code [3][4]), and construct a code for flash memory. The key result in this paper is a generalization of Gray codes from complete permutations [1] to k-partial permutations. In particular, we present Gray codes for arbitrary k-partial permutations that can be generated by push-to-the-top operations - providing efficient information update in flash memories.

Generating all partial permutations has been discussed before (e.g.,[5][6]). Our requirement of push-to-the-top operations is related to universal cycles of k-partial permutations [7][8][9][10]. A *universal cycle of k-partial permutations* is a sequence $(a_1, a_2, \ldots, a_N)$, $N = \binom{n}{k} k!$, $a_i \in \{1, 2, \ldots, n\}$, such that each k-partial permutation is represented by exactly one $(a_{i+1}, a_{i+2}, \ldots, a_{i+k})$ for some $1 \leq i \leq N$. The index additions are computed modulo N. For a given $1 \leq i \leq N$, let $b_1, b_2, \ldots, b_{n-k} \in \{1, 2, \ldots, n\}$ be distinct integers different from $\{a_{i+1}, \ldots, a_{i+k}\}$. Therefore, $P_i = (a_{i+k}, a_{i+k-1}, \ldots, a_{i+1} | b_1, b_2, \ldots, b_{n-k})$ is a permutation. And let $b_0 = a_{i+1}$. Then $a_{i+k+1} = b_j$ for some $0 \leq j \leq n-k$. Let $P_{i+1} = (a_{i+k+1}, a_{i+k}, \ldots, a_{i+2} | b_0, \ldots, b_{j-1}, b_{j+1}, \ldots, b_{n-k})$. From P_i to P_{i+1}, we push $b_j = a_{i+k+1}$ to the top. Thus, traversing the k-partial permutations using push-to-the-top operation is identical to generating a universal cycle (see more details in [11]).

The existence of such universal cycles was proved in [8], however, the proof was nonconstructive. An explicit construction of a cycle for $k = n-1$ was given in [10]. But for other values of k, the construction is still an open problem. In this paper, we will provide constructions of a cycle for all $k \leq n-1$.

We will reduce the problem of generating k-partial permutations to two parts: generating all $\binom{n}{k}$ combinations and all permutations of $k-1$ elements. The latter is again the $(k-2)$-partial permutations out of $k-1$ elements.

Definitions and notations are introduced in Section II and the construction of Gray code for partial rank modulation is explained in Section III. Finally, concluding remarks are made in Section IV.

II. Definitions and Notations

A *k-partial permutation* out of n element is a k-tuple $Q = (p_1, p_2, \ldots, p_k)$ with distinct k elements of a set of size n. The number of possible k-partial permutations is $\binom{n}{k}k!$. The partial permutation is said to be induced from the permutation $P = (p_1, p_2, \ldots, p_n)$, for any ordering of $p_{k+1}, p_{k+2}, \ldots, p_n$. Sometimes we will denote this permutation by $P = (p_1, p_2, \ldots, p_k | p_{k+1}, p_{k+2}, \ldots, p_n)$ to emphasize the first k elements.

A *push-to-the-top operation*, denoted by t_i, is a transition from $(p_1, p_2, \ldots, p_n)$ to $(p_i, p_1, p_2, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)$, for $2 \leq i \leq n$. Throughout this paper, left hand side is considered top, and right hand side is bottom. Notice that if $2 \leq i < k$, then after the operation t_i, there will be a charge gap between the p_{i-1}-th and the p_{i+1}-th cell. As information is stored in the highest k cells, we want to keep these k cells close in charge levels. Especially when large deflation happens, lower charge levels may decrease to none at all, and gap among the first k cells may cause an error. Therefore, in our discussion, we only allow t_i, for $k \leq i \leq n$.

Consider a graph G, whose vertices are permutations of length n. There is a directed edge if there is a push-to-the-top operation t_i from one vertex to another, for $k \leq i \leq n$. So each vertex in G is restricted to $n-k+1$ outgoing edges. An (n, k) *Gray code for partial rank modulations* is a cycle in G whose vertices induce every k-partial permutation once and only once. A Gray code contains $N = \binom{n}{k}k!$ permutations, and we denote them by $\mathbf{P} = (P_1, P_2, \ldots, P_N)$. The question is, does such a cycle exist? We will see in the paper that the answer is yes, and there are indeed a large number of such cycles.

Example 1. *Figure 1(a) shows a $(4, 2)$ Gray code. Only the top two digits in each permutation are of concern. And the push-to-the-top operations are indicated above each edge.*

Define a partition on the partial permutations by the relation $\sim$, where $P_1 \sim P_2$ if the first k elements of P_2 is a cyclic shift of that of P_1. For example, if $k = 3$, $n = 6$, $EQC = \{(1,3,6), (6,1,3), (3,6,1)\}$ is an equivalence class. $\sim$ is obviously an equivalence relation. If $P = (p_1, p_2, \ldots, p_k) \in EQC$, and $p_k = \max\{p_1, p_2, \ldots, p_k\}$, then we choose P as the representative of the equivalence class EQC (unless mentioned otherwise). We will denote EQC by $(p_1, p_2, \ldots, p_k)$. In the previous example, $(1, 3, 6)$ is the representative and we will write $EQC = (1, 3, 6)$. There are $\binom{n}{k}(k-1)!$ equivalence classes.

III. Gray code for partial rank modulation

In this section, we present a construction of (n, k) Gray code for partial rank modulation, for $1 \leq k \leq n-1$. We first construct a tree that contains all the equivalence classes, and then use this tree to generate all the partial permutations.

Notice that starting from any partial permutation, if we do $(k-1)$ times the operation t_k, we can traverse an equivalence class. If we start from some partial permutation in equivalence class EQC_1 and apply operations

$$\underbrace{t_k, \ldots, t_k}_{a \text{ times}}, t_i, \underbrace{t_k, \ldots, t_k}_{k-1 \text{ times}}, t_{k+1}, \underbrace{t_k, \ldots, t_k}_{k-2-a \text{ times}}$$

for $0 \leq a \leq k-2$ and $i > k$, we can still traverse all members of EQC_1. And if the partial permutation after t_i belongs to EQC_2, the above operations also traverse EQC_2. In another word, inserting EQC_2 does not affect the completion of EQC_1. We call the above operation *insertion*. In addition, if $a = k-1$ in the above operation sequence, we omit the last operations t_{k+1} and $(k-2-a)$ times of t_k. And we traverse EQC_1, followed by traversing EQC_2. We consider this case as insertion, too. The permutations circled by dashed line in Figure 1(a) illustrates this notion, where $EQC_1 = (2, 4)$ is inserted by $EQC_2 = (2, 3)$.

We will draw insertions in an *insertion tree*. The nodes of the tree are equivalence classes. An edge means the child is inserted in the parent. Notice the parent and the child differ in only one digit for their representatives. We can insert at most k classes to EQC_1, each changing distinct digit of EQC_1. The cycle in Figure 1(a) is generated by the insertion tree in Figure 1(b). The following lemma is similar to the cycle merging technique in [9], however, the ordering of the lower $n-k$ elements matters in our case.

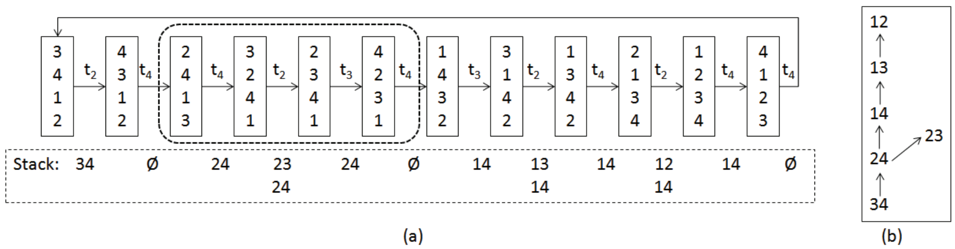

Fig. 1. (a) A cycle for 2-partial permutations out of 4. The corresponding stack storing equivalence classes are shown in the lower dashed box. (b) The insertion tree generating the cycle in (a).

Lemma 2. *Any insertion tree of k elements out of n that contains each equivalence class once and only once will generate an (n,k) Gray code. Such an insertion tree is called a generating tree.*

Proof: We will first prove that a generating tree will induce a path of all k-partial permutations.

Claim: an insertion tree will induce a path of all members of the equivalence classes of the tree (possibly with duplicates if the tree has duplicated nodes). Moreover, the first permutation of the path can be $(p_1,p_2,\ldots,p_k|p_{k+1},\ldots,p_n)$, for any $(p_1,p_2,\ldots,p_k)$ in the equivalence class of the root, and any $X = (p_{k+1},\ldots,p_n)$ as the lower $n-k$ elements.

Use induction on the depth. Base case is one single node, which is true by doing $k-1$ times t_k. For a tree with depth of D, the sub-trees of the root have depth of at most $D-1$ and, by induction, generate paths of all members of their equivalence classes. Let the root be $(p_1,p_2,\ldots,p_k)$ (p_k is not necessarily the largest). For one sub-tree, suppose its root is $(p_1,p_2,\ldots,p_{i-1},p_i',p_{i+1},\ldots,p_k)$, then we can construct a path as follows: $(p_1,p_2,\ldots,p_k|X_1) \xrightarrow{t_k} \cdots \xrightarrow{t_k} (p_{i+1},\ldots,p_k,p_1,\ldots,p_i|X_1) \xrightarrow{\text{push } p_i'} [(p_i',p_{i+1},\ldots,p_k,p_1,\ldots,p_{i-1}|X_2) \rightarrow \cdots \rightarrow (p_{i+1},\ldots,p_k,p_1,\ldots,p_{i-1},p_i'|X_3)] \xrightarrow{\text{push } p_i} (p_i,p_{i+1},\ldots,p_k,p_1,\ldots,p_{i-1}|X_4) \xrightarrow{t_k} \cdots \xrightarrow{t_k} (p_2,\ldots,p_k,p_1|X_4)$, for some X_1,X_2,X_3,X_4 as the lower elements of the permutations. The sub-path in the square brackets corresponds to the sub-tree. If $i=1$, the part after the square brackets is omitted. This path goes through the equivalence classes in the sub-tree and the root completely. Similarly, we can insert other sub-paths into this path. So the claim is true.

Now as a generating tree contains each equivalence class exactly once, we have a path passing every partial permutation exactly once. Next, noticing that such a path satisfies the condition in Lemma 2 in [11], an (n,k) Gray code (a cycle) can be generated. ■

This Gray code in Lemma 2 works like a stack. That is, when we go to the current node m, put it in the stack. If every element in the equivalence class of m is passed through, remove m from the stack. Next, go to a child of m, and treat the child as the current node. If the current node has been traversed, and has no children, then pop a node from the stack and go to that node. For example, in Figure 1(a), the stack after each partial permutation is written in the lower dashed box.

We will give a special class of generating trees based on combinations, where the elements are ordered increasingly. Let $(c_1,c_2,\ldots,c_n)$ be a vector with distinct values and $c_{p_1} > c_{p_2} > \cdots > c_{p_n}$, then define its *reversed rank* as $(p_n,p_{n-1},\ldots,p_1)$. Suppose we have an insertion tree T_1 that contains each combination of k out of n once, and denote a node in T_1 by $(p_1,p_2,\ldots,p_k)$, $p_1 < p_2 < \cdots < p_k$. Suppose the root of the tree is $(n-k+1,n-k+2,\ldots,n)$. Define σ_1 to be the identity permutation on $\{1,2,\ldots,k-1\}$. Let σ_2 be a permutation of $\{1,2,\ldots,k-1\}$. And for an integer x, use σ_2+x to denote the sequence $(\sigma_2(1)+x,\sigma_2(2)+x,\ldots,\sigma_2(k-1)+x)$. If we can transit from a node in T_1 to the node $(\sigma_2+(n-k),n)$, then by the same structure as T_1, except that we change each $(p_1,p_2,\ldots,p_k)$ in T_1 to $(p_{\sigma_2(1)},p_{\sigma_2(2)},\ldots,p_{\sigma_2(k-1)},p_k)$ in T_2, we get another insertion tree T_2 and the dual (the inverse permutation) of the reversed rank of each node in T_2 is (σ_2,k). After that, we transit from T_2 to T_3 with permutation σ_3, and each node in T_1 is replaced by $(p_{\sigma_3(1)},p_{\sigma_3(2)},\ldots,p_{\sigma_3(k-1)},p_k)$, and so on.

Theorem 3. *If there is an insertion tree T_1 that contains every combination of k out of n once, and a sequence of valid transitions from previous trees (as described above), $\sigma_1,\sigma_2,\ldots,\sigma_{(k-1)!}$, traversing all permutations on $\{1,2,\ldots,k-1\}$, then $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_{(k-1)!}$ is a generating tree of k-partial permutations.*

Proof: Recall that we represent the equivalence classes with $(p_1,\ldots,p_k)$ where $p_k = \max\{p_1,\ldots,p_k\}$. For any $p_1 < p_2 < \cdots < p_k$, and any equivalence class $(p_{\sigma(1)},p_{\sigma(2)},\ldots,p_{\sigma(k-1)},p_k)$, there exists exactly one $1 \le i \le (k-1)!$, such that $\sigma = \sigma_i$ and this equivalence class appears exactly once in the tree T_i. ■

Now the problem of finding Gray code for partial rank modulation reduces to finding T_1 and $\sigma_1,\sigma_2,\ldots,\sigma_{(k-1)!}$.

Construction 4. *(Construction of T_1) The root $(n-k+1,n-k+2,\ldots,n)$ has one child $(n-k,n-k+2,\ldots,n)$, and the node $(1,2,\ldots,k)$ has a parent*

$(1, 2, \ldots, k-1, k+1)$. *Starting from the root, and following the connection rules below for all the other nodes, we will get a* T_1.

1) *A node* $(p_1, p_2, \ldots, p_{k-n+t-1}, t, t+1, \ldots, n)$, *for* $1 \le p_{k-n+t-1} < t-1$ *and* $n-k+2 \le t \le n$, *is connected to the at most 3 nodes:*

Parent $(p_1, p_2, \ldots, p_{k-n+t-1} + 1, t, t+1, \ldots, n)$
Child$_1$ $(p_1, p_2, \ldots, p_{k-n+t-1} - 1, t, t+1, \ldots, n)$ *if* $p_{k-n+t-1} - 1 > p_{k-n+t-2} \ge 1$
Child$_2$ $(p_1, p_2, \ldots, p_{k-n+t-1}, t-1, t+1, \ldots, n)$

2) *Otherwise, a node* $(p_1, p_2, \ldots, p_k)$ *is connected to at most two nodes:*

Parent $(p_1, p_2, \ldots, p_k + 1)$
Child $(p_1, p_2, \ldots, p_k - 1)$ *if* $p_k - 1 > p_{k-1}$

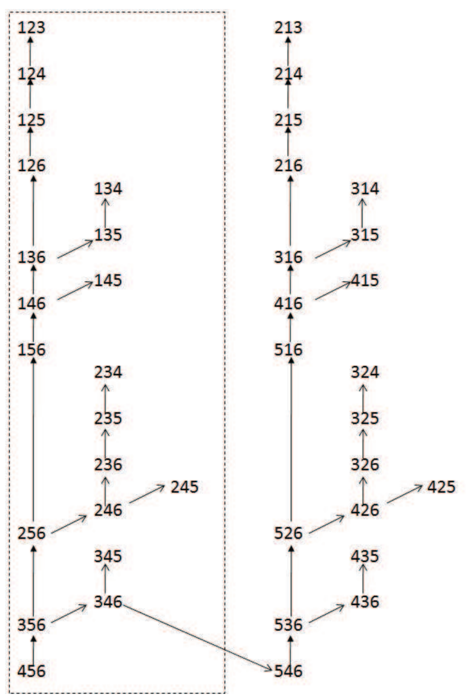

Fig. 2. A generating tree of 6 out of 3 based on Construction 4 and cycle $12 \xrightarrow{t_2} 21 \xrightarrow{t_2} 12$

Figure 1(b) is one example of Construction 4. Another example of 6 out of 3 is shown in Figure 2. T_1 is the left part of the tree in dashed line. T_1 is similar to listing combinations lexicographically.

Theorem 5. *Construction* 4 *forms a tree* T_1 *that contains every combination of* k *out of* n.

Proof: The construction already includes every combination. So we need to prove that T_1 is a connected graph with no cycles.

Starting from any node in T_1, and tracing back the parent, is equivalent to increasing the k-th digit to the maximum, and then increasing the $(k-1)$-th digit to the maximum, and so on. All of these backward paths will end at $(n-k+1, n-k+2, \ldots, n)$. Therefore, graph T_1 is connected. Moreover, as a node is always smaller than its parent in lexicographical order, there is no cycle in T_1. Hence T_1 is a tree. ■

Another possible T_1 is a single line. There are several different constructions in the literature. For example, homogeneous scheme and near-perfect scheme in [12].

For construction of $\sigma_1, \ldots, \sigma_{(k-1)!}$, we will assume $k \ge 3$, because otherwise only T_1 itself is the generating tree. We first find which transitions are allowed from sub-tree T_t to T_{t+1}. Consider the node in T_t, $a = (p_1, p_2, \ldots, p_{i-1}, n-k, p_{i+1}, \ldots, p_{k-1}, n)$, where $\{p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_{k-1}\} = \{n-k+1, n-k+2, \ldots, n-1\} \setminus \{y\}$ for some $n-k+1 < y \le n-1$. This node exists since $n \ge k+1$. We can now define a child of a as $b = (p_1, p_2, \ldots, p_{i-1}, y, p_{i+1}, \ldots, p_{k-1}, n)$. Let $x = y - n + k$ and b be the root of T_{t+1}. Since (σ_t, k) is dual of reversed rank of a and (σ_{t+1}, k) is dual of reversed rank of b, we can see that $\sigma_{t+1} = \alpha_x \circ \sigma_t$, where $\alpha_x = (x, 1, 2, \ldots, x-1, x+1, \ldots, k-1)$. Here $\circ$ is the composition of permutations and is computed from right to left. It can be seen that the child of a in Construction 4 changes a different digit of a, compared to b. Hence, α_x is a valid transition from tree T_t to T_{t+1}.

Permuted by α_x, the lowest position in σ_t becomes the x-th lowest position in σ_{t+1}. In another word, from the dual of σ_{t+1} to the dual of σ_t, we push the x-th element to the top.

Construction 6. *(Construction of* $\sigma_1, \ldots, \sigma_{(k-1)!}$*) Suppose* $\mathcal{C}$ *is a cycle of permutations of* $\{1, 2, \ldots, k-1\}$. *First take the dual of each permutation in* $\mathcal{C}$. *Then replace each push-to-the-top operation* t_x *with* α_x *and reverse the edge direction. And thus we will get a cycle of* $\sigma_1, \sigma_2, \ldots, \sigma_{(k-1)!}$, *generated by operations* α_x, $1 < x \le k-1$. *Starting from the identity permutation* σ_1, *and ignoring the last edge, we get a path of* $\sigma_1, \sigma_2, \ldots, \sigma_{(k-1)!}$.

$\sigma_1, \sigma_2, \ldots, \sigma_{(k-1)!}$ can be easily generated once we have recursively generated $(k-2)$-partial permutations out of $k-1$ (or permutations on $k-1$ elements) using the constructions above. We can also use the cycle of permutations in [1][10]. For example, if $k = 4$, then Construction 4 will form a cycle $\mathcal{C}$ on 3 elements using the operation sequence $t_2, t_3, t_3, t_2, t_3, t_3$. And the push-to-the-bottom sequence is $231 \xrightarrow{t_2} 321 \xrightarrow{t_3} 132 \xrightarrow{t_3} 213 \xrightarrow{t_2} 123 \xrightarrow{t_3} 312 \xrightarrow{t_3} 231$. The corresponding sigma cycle is $312 \xleftarrow{\alpha_2} 321 \xleftarrow{\alpha_3} 132 \xleftarrow{\alpha_3} 213 \xleftarrow{\alpha_2} 123 (\xleftarrow{\alpha_3}) 231 \xleftarrow{\alpha_3} 312$. Deleting the arrow in brackets, we get a path.

Having constructed T_1 and $\sigma_1, \ldots, \sigma_{(k-1)!}$, we are able to draw a generating tree. Figure 2 shows a generating tree of 6 out of 3. The sequence σ_1, σ_2 is $12 \xrightarrow{\alpha_2} 21$, and the transition node from T_1 to T_2 is $(3, 4, 6)$.

Knowing a generating tree and the current partial permutation, how shall we decide the operation for next step? Define w_m to be the *position of the newly changed digit* from the parent to the node m (counted from top to bottom). Let $m = (p_1, \ldots, p_k)$, then its parent defers with m only at p_{w_m}. For the root, w_{root} can be any integer between 1 and k. But we will assign w_{root} as k. For example, in Figure 2, $(1, 2, 4)$ is the parent of the node $m = (1, 2, 3)$, and the 3rd digit is changed from 4 to 3, so $w_m = 3$, $p_{w_m} = p_3 = 3$. The following algorithm recursively computes α_x and the next permutation:

Algorithm 7. *($G(n, k)$)*

Given a permutation, take the top k elements $(q_1, q_2, \ldots, q_k)$, and suppose $q_l = \max_{i=1}^{k} q_i$. Let $j - 1$ be the number of elements appearing after q_l. Order the top k elements increasingly as $m = (p_1, p_2, \ldots, p_k)$.

1 *If $m = (p_1, p_2, \ldots, p_{i-1}, n - k, p_{i+1}, \ldots, p_{k-1}, n)$, where $\{p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_{k-1}\} = \{n - k + 1, n - k + 2, \ldots, n - 1\} \setminus \{y\}$ for some $n - k + 1 < y \leq n - 1$, and if $j - 1 \equiv k \bmod k$, compute σ and α_x. σ is the dual of the reversed rank of $(q_{l+1}, q_{l+2}, \ldots, q_k, q_1, \ldots, q_{l-1})$, and α_x is obtained recursively by the reverse algorithm of $G(k - 1, k - 2)$ on dual of σ. If $x = y - n + k$, push x to the top (Transit to the next sub-tree).*

2 *Else, if $m = (n - k + 1, n - k + 2, \ldots, n)$ and $j - 1 \equiv k \bmod k$, then push $n - k$ to the top (Transit to the previous sub-tree).*

3 *Else, first compute parameters: if $m = (p_1, p_2, \ldots, p_{k-n+t-1}, t, t + 1, \ldots, n)$, for $1 \leq p_{k-n+t-1} < t - 1$ and $n - k + 2 \leq t \leq n$, then $w_m = k - n + t - 1$, $w_y = k - n + t - 1$ (only if $p_{k-n+t-1} - 1 > p_{k-n+t-2} \geq 1$) and $w_{y'} = k - n + t$, and $p^{(y)}_{w_y}$ is $p_{k-n+t-1} - 1$ (only if $p_{k-n+t-1} - 1 > p_{k-n+t-2} \geq 1$), and $p^{(y')}_{w_{y'}} = t - 1$, respectively; else, $w_m = w_y = k$, and $p^{(y)}_{w_y} = p_k - 1$.*
   - *If $p_{k-1} + 1 = p_k \neq n$ (m has no children): if $j \equiv w_m + 1 \bmod k$, then push $p_k + 1$ to the top; else, do t_k.*
   - *Else (m has children): if $j \equiv w_y + 1$ (or $j \equiv w_{y'} + 1$) mod k for any child, push $p^{(y)}_{w_y}$ (or $p^{(y')}_{w_{y'}}$) to the top; else if $j \equiv w_m + 1 \bmod k$, then push $t - 1$ to the top if $m = (p_1, p_2, \ldots, p_{k-n+t-1}, t, t + 1, \ldots, n)$ for $1 \leq p_{k-n+t-1} < t - 1$, and $n - k + 2 \leq t \leq n$, otherwise push n to the top; else do t_k.*

Figure 1(a) is an example of the above algorithm with $n = 4$, $k = 2$. For instance, if the current permutation is $(1, 4|2, 3)$, then $j - 1 = 0$ and $m = (p_1, p_2) = (1, 4)$. σ is identity. $t = p_2 = 4$, $w_m = k - n + t - 1 = 1$, and $w_{y'} = k - n + t = 2$. As $j = w_{y'} + 1$, We push $p^{(y')}_{w_{y'}} = t - 1 = 3$ to the top. The next permutation is $(3, 1|4, 2)$.

Even though the algorithm $G(n, k)$ is recursive, we can show the number of recursions is less than 1 on average [11]. Hence, the average computational complexity of $G(n, k)$ is only dependent on the non-recursive operations. In particular, dependent on ordering the top k elements increasingly as m. Thus, the complexity for Algorithm 7 is $O(k \log k)$.

IV. Conclusion

In this paper we proposed a generalization of rank modulation - k-partial rank modulation - for flash memories. This coding scheme allows less decoding cost while sacrificing some information capacity. We presented a Gray code that traverses all k-partial permutations out of n elements, using push-to-the-top write operations, which is designed specifically for flash cells. The permutation cycle is reduced first to a generating tree with cyclic shift classes as nodes. In the generating tree, adjacent nodes differ in one digit. And then it is further simplified to combination generation and permutation generation. Each step in the constructed cycle can be determined solely by the current permutation.

Several problems are still open in this topic. For instance, how to construct insertion trees with minimum, maximum, or evenly distributed gap in the lower $n - k$ cells. Also, whether it is possible to use the lower cells as redundancy to protect the top k cells against errors.

Acknowledgment

This work was supported in part by the NSF Grant ECCS-0802107 and in part by an NSF-NRI award.

References

[1] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck. Rank modulation for flash memories. *Information Theory, IEEE Transactions on*, 55(6):2659 –2673, June 2009.

[2] J. E. Brewer and M. Gill. *Nonvolatile memory technologies with emphasis on flash.* Wiley-IEEE, 2007.

[3] A. Jiang, V. Bohossian, and J. Bruck. Floating codes for joint information storage in write asymmetric memories. *Proc. IEEE ISIT*, 2007.

[4] M. Mitzenmacher, Z. Liu, and H. Finucane. Designing floating codes for expected performance. *Proc. of the Forty-Sixth Annual Allerton Conference*, 2008.

[5] G. Ehrlich. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *Journal of the ACM (JACM)*, 20(3):500–513, 1973.

[6] N. Dershowitz. A simplified loop-free algorithm for generating permutations. *BIT Numerical Mathematics*, 15(2):158–164, 1975.

[7] R. Graham F. Chung, P. Diaconis. Universal cycles for combinatorial structures. *Discrete Mathematics*, 110(1-3):43–59, 1992.

[8] B. W. Jackson. Universal cycles of k-subsets and k-permutations. *Discrete mathematics*, 117(1-3):141–150, 1993.

[9] J.R. Johnson. Universal cycles for permutations. *Discrete Mathematics (in press)*, 2008.

[10] F. Ruskey and A. Williams. An explicit universal cycle for the (n-1)-permutations of an n-set. In *Talk at Napier Workshop*. Citeseer, 2008.

[11] Z. Wang and J. Bruck. Partial rank modulation for flash memories. available at $http://www.paradise.caltech.edu/etr.html$.

[12] D. E. Knuth. *The art of computer programming, Volume 4, Fascicle 3.* Addison-Wesley, 2005.